

AAASeed - Net Requests

Documentation for the net_requests Class

The `net_requests` class in the AAA framework provides a set of functions to make HTTP requests, handle JSON responses, and easily authenticate with all common methods. This documentation will provide an overview of the class and its capabilities. **The functionality provided here gives AAASeed the power to access a large portion of the internet and interact with nearly all APIs.**

Overview:

The `net_requests` class is responsible for handling asynchronous HTTP requests such as GET, POST, PUT, PATCH, DELETE, and downloading files. It uses the [CPR](#) library for making HTTP requests and RapidJSON to handle JSON data, with a custom parsing function to perform conversion from JSON to Lua tables for ease of access. Additionally, it automatically handles HTTPS authenticated requests and has functions for providing username/password authentication in Basic, Digest, or NTLM mode, as well as for OAuth - Bearer Token (one of the most common for interacting with APIs these days). This class is exposed to Lua through the global `net_requests` table (`aaa.net_requests`), defined in `net_requests_lua.cpp`.

This class prioritizes **simplicity** and **ease of use** to make this functionality as accessible as possible for artists looking to build installations with AAASeed that interact with APIs. All requests are handled asynchronously outside of Lua in a completely non-blocking manner, ensuring that requests or downloads with high latency create zero impact on the other processes and visuals running in AAASeed.

The high-level user flow for interacting with the `net_requests` class is as follows:

1. Make a request with the desired HTTP verb.
2. Receive the ID of the request.
3. Check as frequently as desired to determine whether the request was completed or not, and if it was successful or not.
4. If the request was successful, parse the result and use it.

This leaves it up to the needs of the specific application to decide how often to check the results of a request in-progress - some applications may want to check this every update frame to get the result as soon as possible, while others may only care to check at a longer interval. In any case, the “checking” process to see if a request is done creates virtually zero overhead and has no impact on rendering time, even if checked every frame.

Multiple requests may be executed simultaneously via the same functions - the only thing that a Lua user needs to pay attention to is keeping tabs on the returned request IDs to ensure that they're checking for the completion and results of the correct request.

To ease this process (depending on the application), a function is also provided for enumerating the request IDs and their completion status as a Lua table. This allows for a Lua function to be simply written to iterate over this table and fetch the results of any requests that have been completed.

Notably, for simplicity, the act of retrieving a response to a request deletes the response from the response map contained within `net_requests`. This means that only one MEU per application should be responsible for obtaining the response to any single request - otherwise, requests will appear to disappear if a second MEU is attempting to retrieve the result of the same request as another. This may be an area to modify in the future depending on user requirements, but is expected to be acceptable and preferred behaviour for now to prevent accidental reading of responses for old requests.

API Reference

All functions are provided in the `aaa.net_requests` table.

Authentication

Each of these functions sets the credentials for the corresponding authentication mode and automatically enables the mode for **all** subsequent requests.

- `set_basic_auth(username, password)`
- `set_digest_auth(username, password)`
- `set_ntlm_auth(username, password)`
- `set_oauth(token)`

Authentication can be temporarily disabled and re-enabled later (without having to pass in credentials and mode again) by using:

- `disable_auth()`
- `enable_auth()`
 - Returns 1 if successful, 0 if mode/credentials were not set previously

Credentials can be cleared using:

- `clear_auth()`

Requests

Methods below are unauthenticated by default. Calling any of the `set` functions in the Authentication section will enable authentication for all requests unless `disable_auth()` or `clear_auth()` is called. HTTPS is supported automatically.

All functions here return a `request_id`.

- `request_get(url)`
- `request_post(url, payload)`
- `request_put(url, payload)`
- `request_delete(url)`
- `request_patch(url, payload)`
- `request_download(url)`

Receiving and Parsing Responses

It's left up to the designer to decide the frequency at which to check the status of a request and obtain its response.

- `get_request_result_if_done(request_id)`
 - `request_id` is the value returned by all of the `request` methods in the section above. This allows for specific requests to be queried without touching others. If the request has completed, **the response will be retrieved and the request will be cleared from the list.**
 - Responses are a table with the following members:
 - `status_code`
 - HTTP status code of the response
 - `text`
 - Response body of the request
 - `url`
 - URL of the request
 - `elapsed`
 - Time taken for the request to complete in seconds
- `get_request_list()`
 - Returns a table with keys containing `request_id` of all non-cleared requests and values containing `1` or `0` as to whether a response is available or not. **This allows for checking request status without clearing requests from the list.** A designer can then call `get_request_result_if_done(request_id)` to obtain the response.
- `json_to_table(text)`
 - Parses a JSON given as a string `text`, returning it as a Lua table.

Examples

The `TutoLuaNetRequest` MEU is available in AAASeed as a basic demonstration of how to integrate `aaa.net_requests` into a MEU.

Code Snippets

Let's assume we've called a function that makes a request using the following:

```
-- This url returns the temperature series for the following day in Toronto, Canada
in JSON format
local url =
"https://api.open-meteo.com/v1/forecast?latitude=43.7001&longitude=-79.4163&hourly=te
mperature_2m&forecast_days=1"
-- self.request_id is saved as a property of the MEU
self.request_id = aaa.net_requests.request_get(url)
```

Either in the `meu:update()` function or elsewhere, we can check for the request being done:

```
-- Ensure request_id is valid
if (self.request_id ~= nil) then
    -- See if the request is done (false if not, response object if done)
    local res = aaa.net_requests.get_request_result_if_done(self.request_id)
    if (res ~= false) then
        -- Ensure request received a successful response
        if (res.status_code == 200) then
            -- Parse the response as JSON and convert to Lua table
            self.response = aaa.net_requests.json_to_table(res.text)
        end
        -- Reset request_id to nil so we stop checking until another request is made
        self.request_id = nil
    end
end
```

Alternatively, if we had a lot of requests to check and we needed to check all of them:
(We'll assume that we have a `self.responses` table for the MEU which we use to store all the responses to our requests)

```
-- Get the list of requests that haven't been completed or checked yet
local request_list = aaa.net_requests.get_request_list()
-- Iterate over the id and completion status of each request
for id, done in pairs(request_list) do
    -- If the request is complete...
    if (done == 1) then
        -- Get the response
        local res = aaa.net_requests.get_request_result_if_done(id)
        if res ~= false then
            -- Parse the body as a JSON and convert to Lua table
            local response_table = aaa.net_requests.json_to_table(res.text)
        end
    end
end
```

```
        -- Insert that table into our self.responses table for later use
        table.insert(self.responses, response_table)
    end
end
end
```

Videos

-  Open Meteo Weather API Demo.mp4